Exemplo de documentação para os TPs

1 Introdução

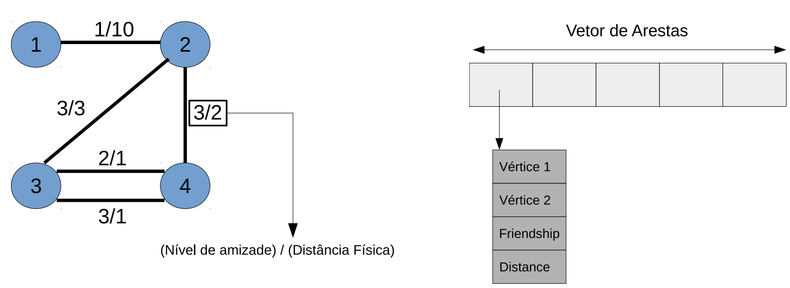
O objetivo deste trabalho é a resolução do problema de recomendação de amigos na *Orkato*, uma das maiores redes sociais no Brasil. As únicas informações disponíveis são os elos de amizades entre todos os pares de amigos na rede. Este elo de amizade contêm a distância física entre o par (Distance) e um valor que determina o nível de amizade entre eles (Friendship). Dessa forma, a equipe chegou a uma conclusão que a melhor forma de resolver o problema é determinar um subgrafo que conecte todos usuários da rede de tal forma que a função de qualidade seja maximizada. Essa função de qualidade é descrita pela equação 1.

$$Qualidade = \frac{\sum_{i=1}^{E} Friendship_i}{\sum_{i=1}^{E} Distance_i} \tag{1}$$

Caso não seja possível entrar algum subgrafo, visto que a rede pode ser desconectada, é necessário que a solução relate este fato para que a equipe recomende amigos de forma a poder tornar o grafo conectado.

2 Modelagem do Problema

A visualização gráfica do problema pode ser vista como um multi-grafo não direcionado onde cada vértice representa um usuário da rede e a aresta representa o elo entre eles. Este elo contêm duas informações: a distância física (Distance) e o nível de amizade entre os vértices (Friendship). A figura 1(a) mostra um exemplo deste grafo. A figura 1(b) mostra a estrutura de dados utilizada para armazenar o grafo.



(a) Representação do problema por meio de grafos.

(b) Estrutura de dados utilizada para armazenar o grafo.

Figure 1: Modelagem do problema utilizando grafos.

Para encontrar o subgrafo que maximiza a função de qualidade e mantenha todos os vértices conectados, podemos fazer uma pequena modificação na função de qualidade, conforme segue abaixo:

$$Qualidade = \frac{\sum_{i=1}^{E} Friendship_i}{\sum_{i=1}^{E} Distance_i} \geq R \tag{2}$$

A ideia geral é apresentar uma solução R e verificar se, no grafo, existe um subgrafo que consiga gerar uma solução maior que a apresentada e que mantém todos os vértices conectados. Se sim, a solução consegue ser maior e apresentamos uma solução R_1 tal que $R_1 > R$. Caso contrário, apresentamos uma solução R_2 tal que $R_2 < R$. Para verificar se tal subgrafo existe, podemos remotar função de qualidade da equação 2 e gerar a equação 3.

$$Qualidade = \sum_{i=1}^{E}(Friendship_i) - R * \sum_{i=1}^{E}(Distance_i) \geq 0 \tag{3}$$

Pode-se derivar da equação 3 que, se uma aresta é escolhida para a solução final, ela acrescenta/diminui a função de qualidade em um valor Y dado pela equação 4. Dessa forma, podemos reescrever o peso de todas as arestas do grafo de acordo com a equação 4 e verificar se existe um subgrafo que mantém todos os vértices conectados e cuja soma de todos os pesos das arestas escolhidas resulta em um valor maior ou igual a 0. Se tal subgrafo existir, então existe uma solução R_1 tal que $R_1 > R$. Se este subgrafo não existe, então a solução é menor que R.

$$Y = (Friendship_i) - R * (Distance_i) \tag{4}$$

Para checar se existe um subgrafo com as duas propriedade citadas, podemos utilizar o algoritmo de Kruskal para gerar uma Árvore Geradora Máxima, de forma a garantir que o grafo se manterá conectado e terá o maior valor possível. Entretanto, o algoritmo de Kruskal pode descartar arestas que aumentam a solução final se estas formam ciclos. Observe que, se uma aresta possui custo positivo após a reescrita dos pesos utilizando a 4, ela só irá aumentar o lado direito da equação 3. Logo, devemos colocar essa aresta na solução, mesmo que ela forme ciclos visto que ela aumentará a função de qualidade. Dessa forma, dada uma solução R qualquer, podemos verificar se existe uma solução R_1 tal que $R_1 > R$ aplicando o algoritmo 1. Para simplificar o algoritmo, a função "Forma_Ciclo(aresta)" substituiu a estrutura de dados *UnionFind*, que foi utilizada na implementação.

Algorithm 1: *Kruskal* (Grafo G, Solucao R)

```
// Renomeia o peso das arestas de acordo com a equação 4
G' = renomeia_peso_arestas(G,R);
// Ordena as arestas em ordem decrescente para realizar o algoritmo
// de Kruskal de forma a encontrar a Árvore Geradora Máxima.
arestas_ordenadas = ordena(G'.arestas());
custo = 0;
// Realiza o algoritmo de Kruskal, com a modificação de incluir arestas com
// custo positivo.
for aresta em arestas_ordenadas do
    if aresta.custo ≥ 0 || not Forma_Ciclo(aresta) then
        custo += aresta.custo;
Retorna (custo >= 0);
```

Logo, o algoritmo 1 fornece um modo prático de verificar se uma solução R é alcançável ou não. A segunda parte do problema é: como escolher o R? A equação 5 fornece uma intuição importante usada para deduzir a restrição do espaço de busca mostrado pela equação 6. A equação 5 mostra que, para a função de qualidade descrita pela equação 1, é necessário fornecer uma razão (Friendship/Distance) que seja maior que a razão atual.

$$\frac{x+a}{y+b} > \frac{x}{y} \rightarrow xy + ay > xy + by \rightarrow \frac{a}{b} > \frac{x}{y} \tag{5}$$

Observe que, se a aresta com a maior razão $ratio_{max}$ no grafo for escolhida para a solução final, nenhum valor será maior que $ratio_{max}$ e a solução final será menor ou igual que $ratio_{max}$. O inverso também é verdade: se a aresta com menor razão for escolhida $ratio_{min}$, então a solução só poderá ser maior ou igual a $ratio_{min}$. Se escolhermos qualquer conjunto de arestas que não inclui $ratio_{max}$ e $ratio_{min}$, haverá um $ratio'_{max}$ e um $ratio'_{min}$ tal que $ratio_{max} \geq ratio'_{max}$ e $ratio_{min} \leq ratio'_{min}$ Isso resulta na equação 6, que faz com que o espaço de busca por R seja restringido.

$$min_{\forall j \in E}(\frac{Friendship_j}{Distance_j}) \leq \frac{\sum_{i=1}^{E} Friendship_i}{\sum_{i=1}^{E} Distance_i} \leq max_{\forall k \in E}(\frac{Friendship_k}{Distance_k}) \quad (6)$$

Logo, podemos fazer o problema fazendo uma busca binária no espaço de solução de forma a encontrar o maior valor possível de R. O algoritmo 2 sintetiza o algoritmo completo que permite encontrar o maior valor de R com um erro de até 3 casas decimais.

Algorithm 2: *BuscaBinária* (Grafo G)

```
maior = G.razao_min();
menor = G.razao_max();
while absoluto(maior - menor) ≥ 0.001 do
    meio = (maior+menor) /2 ;
    // Verifica se o algoritmo de Kruskal modificado retorna
    // se é possível resolver o problema.
    consegue_resolver = Kruskal(G,meio);
    if consegue_resolver then
        menor = meio;
    else
        maior = meio;
Retorna menor;
```

Porém, só falta um detalhe para obtermos a modelagem completa para resolver o problema: verificar se o grafo é desconectado. Em caso afirmativo, reportamos que ele é desconectado. Caso contrário, fazemos a busca binária no espaço de solução limitado pela equação 6 para encontrar a solução exata. O algoritmo 3 apresenta o peça final para a conclusão da modelagem.

Algorithm 3: *EstáConectado*(Grafo G)

```
if Esta_Conectado then
    imprime BuscaBinária(G);
else
    imprime -1.000;
```

3 Análise Teórica do Custo Assintótico

Nesta seção, será apresentada a análise do custo teórico de tempo e de espaço.

3.1 Análise Teórica do Custo Assintótico de Tempo

Podemos analisar a análise do custo assintótico avaliando cada um dos três algoritmos apresentados:

- O algoritmo 1 realiza o algoritmo de Kruskal avaliando todas as arestas e não até encontrar |V|-1 primeiras arestas da ordenação que não formam ciclo. Porém, este é o pior caso do algoritmo de Kruskal, quando este tem que percorrer até a última aresta. O custo assintótico do algoritmo de Kruskal é $O(|E|\log|E| + |E|\alpha(|V|))$ onde E são as arestas do grafo, V são os vértices e $\alpha(|V|)$ é a função de ackermann que cresce muito lentamente. Como a repesagem das arestas é feita com custo $O(|E|)$, pois precisamos repassar cada aresta uma única vez, o custo total do algoritmo 1 é $O(|E|\log|E| + |E|\alpha(|V|) + |E|)$. Dessa forma, o maior custo ainda continua sendo a ordenação e o custo assintótico do algoritmo 1 é $O(|E|\log|E|)$

- O algoritmo 2 realiza uma busca binária no espaço de solução determinado pela equação 6 executando, em cada iteração da busca, o algoritmo 1. O número de buscas é proporcional a $\log_2(W)$, onde W é definido como a diferença entre $ratio_{max}$ e $ratio_{min}$, que é o tamanho do espaço de busca. Entretanto, temos que lembrar que existe uma precisão associada na busca. Como a precisão é de 3 casas decimais, o espaço de busca é como se fosse multiplicado por 10.000, ou seja, W=10.000*W. Dessa forma, o custo total do algoritmo 2 é $O(100 * \log_2(W) * |E|\log|E|)$.

- O algoritmo 3 faz apenas uma verificação da conectividade do grafo. Caso o grafo seja desconectado, é retornado apenas um valor indicando a situação. Caso contrário, é executado o algoritmo 2. Pode-se usar a estrutura de dados *UnionFind* para verificar se o grafo é desconectado. O algoritmo repassa cada aresta uma única vez, atualizando os grupos dos vértices da extremidade na estrutura de dados *UnionFind*. Ao final, pegamos o grupo de um vértice qualquer e verificamos se o grupo de algum vértice é diferente. Se for, então o grafo é desconectado. Assim, o custo total para verificar se há ciclos é $O(|E|\alpha(|V|) + |V|)$, que resulta em $O(|E|\alpha(|V|))$. Logo, o custo total do algoritmo é a soma do custo verificação de ciclos e do custo do algoritmo 2. Assim, o custo é $O(|E|\alpha(|V|) + 100 * \log_2(W) * |E| \log |E|)$. Como o termo dominante é o algoritmo 2, o custo total do algoritmo é $O(100 * \log_2(W) * |E| \log |E|)$.

Dessa forma, conclui-se que o custo total de tempo do algoritmo é $O(100 * \log_2(W) * |E| \log |E|)$, com W sendo a diferença entre $ratio_{max}$ e $ratio_{min}$.

3.2 Análise Teórica do Custo Assintótico de Espaço

Para fazer a análise do custo de espaço, focamos nas duas principais estruturas de dados: o grafo e o *UnionFind*. Para este trabalho, o grafo foi modelado como um vetor de arestas. Cada aresta possuindo 4 informações: os vértices da extremidade, o friendship e o distance. As quatro informações são inteiros. Considerando um inteiro de 4 bytes, isso resulta em um custo $O(16 * |E|)$ ou $O(|E|)$. O *UnionFind* utiliza um vetor do tamanho do número de vértices, onde é guardado o grupo de cada vértice. Isso resulta em um custo de $O(|V|)$. Dessa forma, a complexidade de espaço do algoritmo em si é proporcional a soma dessas duas estruturas, ou seja, $O(|V| + |E|)$[1].

4 Análise de Experimentos

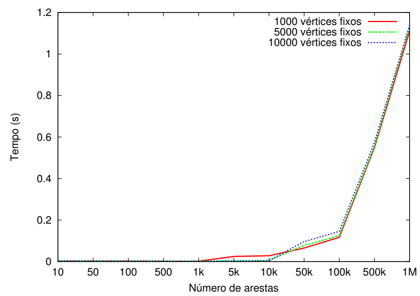
A análise experimental da implementação é mostrada pela figura 2. Para realizar os experimentos, foi feito um gerador de multigrafos sintéticos que tem como saída um grafo, no formato da especificação, com o número de arestas e vértices desejados. Para medir o tempo de execução do código, foi utilizada a biblioteca "<time.h>" para contar o número de clocks necessários para executar o código. Dada a contagem do número de clocks, a conversão para segundos é simples visto que tal biblioteca fornece o número de clocks por segundo através da variável CLOCKS_PER_SEC. Cada instância fornecida pelo gerador foi executada 5 vezes e o tempo de execução foi obtido retirando a média de tempo das 5 execuções. Os testes foram realizados em uma máquina com processdor Core i5 1.6GHz e 4GB de memória ram.

O gráfico 2(a) fornece uma visão do tempo de execução do algoritmo para as bases sintéticas geradas fixando três valores de vértices e variando o número de arestas exponencialmente, ou seja, com um fator multiplicativo de 10. A razão inicial é verificar, de forma geral, o comportamento do algoritmo para um intervalo grande sem precisar gerar muitos pontos. Dessa forma, foram geradas bases com 10, 50, 100, 500, ..., 500.000 e 1 milhão de arestas. Cada uma das três curvas representa um valor fixo de vértices variando o número de arestas de acordo com a variação citada. Como pode ser observado, as três curvas possuem valores bastante similares, mesmo possuindo uma quantidade de vértices de, até, uma ordem de grandeza menor. Retomando o custo teórico assintótico obtido na seção 3.1 que foi $O(100 * \log_2(W) * |E| \log |E|)$, pode-se observar que ele não depende do número de vértices e, dessa forma, os resultados se confirmam.

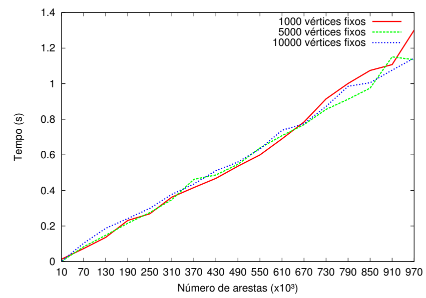
Entrentanto, como a variação do número de arestas foi feito de forma exponencial, os pontos não são representativos o suficiente para mostrarmos que o tempo do algoritmo cresce linearmente em relação ao número de arestas, conforme mostrado no custo teórico

[1] Na implementação, como foram dados os limites do problema, o tamanho dos vetores foi alocado estaticamente. Entretanto, se não houvesse esses limites, a implementação se torna proporcional ao resultado da análise de custo de espaço obtida

assintótico: o termo |E| é multiplicado por dois termos logarítmicos e uma constante. Dessa forma, a figura 2(b) faz o estudo variando o número de arestas no intervalo de 10k a 1 milhão de 20 em 20k, ou seja, fornecemos mais pontos, igualmente distribuídos, para o intervalo. O número de arestas abaixo de 10k é pequeno e quase constante, conforme mostrado na figura 2(a). Logo, pode-se focar apenas no intervalo citado (10k a 1 milhão). O resultado da figura 2(b) mostra exatamente a peça final que encaixa a análise experimental e o custo teórico assintótico: o tempo de execução cresce linearmente com o número de arestas (|E|). Novamente, note que o compoentamento das três curvas, com número de vértices distintos, são bastante similares. Isso reforça a análise feita da 2(a).



(a) Análise experimental variando arestas de 10 a 1M



(b) Análise experimental variando arestas de 10k a 1M

Figure 2: Análise experimental.

Como foram fornecidos os limites do grafo para os testes e na documentação não faz a restrição de utilizar estruturas alocadas dinamicamente, o código funciona utilizando alocações estáticas[2]. Dessa forma, o custo de memória do problema é constante e não precisa ser analisada experimentalmente. Entretanto, o custo teórico assintótico da complexidade de espaço é mostrado na seção 3.2, considerando o caso em que os limites não são conhecidos e as estruturas de dados devem ser alocadas dinamicamente.

5 Conclusão

Neste trabalho, foi resolvido o problema de recomendação de amigos na *Orkato*. O problema foi resolvido modelando-o como grafos e aplicando algoritmos de AGM (Kruskal) e busca binária ao mesmo. A análise de complexidade teórica de tempo foi comprovada através de experimentos que utilizaram entradas grandes suficientes para análisar o comportamento assintótico.

[2] Essa dúvida foi retirada tanto no fórum quanto com o professor da disciplina